Specification Amendments

Amend the paragraphs beginning on line 14 of page 14 and ending on line 4 of page 16, to correct typographical errors as follows:

According to the design of one embodiment, the bit-wise check bits (BCBs) are stored in the rows of the BEDAC 54 as illustrated in Fig. 4, wherein the first row of the BEDAC 54 stores BCB0 for each column of the DRAM 30, the second row of the BEDAC 54 stores BCB1 for each column of the DRAM 30, and so on. As in the WEDAC 52, the error correction algorithm *per se* implemented by the BEDAC 54 can be quite conventional. As is common in such algorithms, and unlike simple parity, each of the bits comprising each code word in the DRAM 30 affects more than one of the respective set of BCBs. For example, according to one possible scheme shown in Fig. 5A, the word-wise code word bits stored in row 0 in the DRAM 30 affect BCB0, BCB1, and BCB2 of the respective bit-wise code word stored in the BEDAC 54, the row 1 code word bits effect BCB0, BCB1 and BCB3, and so on. To implement such a scheme, the BEDAC 54 is coupled to a BEDAC decoder 56 which enables the appropriate BCBs according to the decoding table shown in Fig. 5B, wherein RA[2-0] comprise the same row address select signals provided by the memory controller 46 during an access to the DRAM 30. In accordance with our invention, the BEDAC 54 effectively implements a classic "block error detection and correction" code, not in the traditional word-parallel manner, but bit-serially, *i.e.*, in whatever order the "data" bits happen to appear, including purely random. We refer to our method of operation as "random access error detection and correction" (RAEDAC) to distinguish it from a traditional word-wise error detection and correction algorithm applied to a serial data stream in which the order of bit presentation is predetermined and cannot vary without invalidating the algorithm. Before we show how the BEDAC 54 cooperates with the WEDAC 52 to significantly enhance the ability of the memory system 44 to detect and correct random, multiple, double-bit errors, we will describe the ~~WEDAC 42~~ BEDAC 54 in greater detail.

As shown in Fig. 6, for each column in the DRAM 30, the BEDAC 54 has two sets of five (5) toggle flip-flops or "T-flops", ~~one set~~ for each of the BCBs: a normal set (Tn[0-4]) which is active during normal operation, and a scrub set (Ts[0-4]) which is active only during a scrubbing operation. In such an arrangement, each pair of sets of T-flops are associated with a respective one of the (N+1) bits comprising a word-wise code word. At the end of a double-bit error scrub operation, all of the T-flops in the respective set (*i.e.*, either the normal or scrub set for a given code word bit location) are simultaneously coupled to a corrupt bit select 58, wherein a shared syndrome calculator 60 calculates a syndrome. Each syndrome is then decoded by a respective syndrome decoder 62 in a conventional manner to provide information regarding the number of bits in error (one, two, or even three, depending upon the selected algorithm), and, if only one bit is in error, the row address corresponding to the particular bit in the bit-wise code word that is in error. When considered collectively, the set of all bit-wise syndromes generated by the BEDAC 54 indicate which of the "columns" in the DRAM 30 have bits in error. Similarly, the syndromes generated by the WEDAC 52 indicate which particular "rows" in the DRAM 30 have bits in error.

By combining the word-wise and the bit-wise error information, it is possible in many cases to unambiguously identify those row/column intersections at which the bit errors are present. Thus, as in the simple parity embodiment, all single double-bit word-wise errors can be detected and corrected. In addition, however, the BEDAC 54 can detect and correct all bit-wise single-bit errors, leaving any remaining stacked double-bit errors (now, hopefully, only single-bit word-wise errors) to be cleaned up in a follow-up scrub operation. Furthermore, using the BEDAC 54, single occurrences of double stacked double-bit word-wise errors can be detected and corrected since such a unique error pattern can occur if and only if the bits located at all four of the row/column intersections are in error.

Amend the paragraphs beginning on line 18 of page 16 and ending on line 6 of page 17, to correctly refer to the BCBx check bits as follows:

On the occurrence of a write error or different old and new data bits, a one (1) output from the OR gate 72 and input to AND gate 74 together with a respective one of the BCB~~S~~ signals, BCB~~S~~[i] (described above in conjunction with Fig. 5). When the BCB~~S~~[i] signal is asserted and the output of OR gate 72 is a one (1), AND gate 74 applies to the clock (C) input of the first D latch 64 a one (1), this occurs when there is either a write error or a difference has been detected between the old and new data bits. The AND gate 74 applies a zero (0) to the clock (C) input otherwise. Simultaneously, the BCB~~S~~[i] signal, inverted by an inverter 76, is applied to the clock (C) input of the second D latch 66. In this arrangement, the T-flop is selectively toggled in response to each assertion of the BCB~~S~~[i] signal, depending upon the output of OR gate 72.

To facilitate system initialization, each D-flip-flop has an asynchronous initialization input, I, which is coupled to a Set/Clear (SET/CLEAR) signal, the logic state of which depends upon the location of the respective T-flop in the BEDAC 54, as shown, for the given example, in the left-most column in Fig. 5A. In this example, an odd parity scheme is being used so if an odd number of rows effect the BCB~~S~~ signal the corresponding T-flop is cleared, and if an even number of rows effect the BCB~~S~~ signal, the corresponding T-flop is set. In this way, the BCB~~S~~0 for each column is set, the BCB~~S~~1 for each column is cleared, *etc.* Since the initialization scheme will be determined at design time, the SET/CLEAR signal for each T-flop can be assigned the appropriate logic value in hardware.

Amend the paragraph beginning on line 16 of page 20 and ending on line 7 of page 21, to correctly refer to the BCBx check bits as follows:

Depending upon the severity of a multi-bit error, the sequencer 78 may not be required to scrub the entire parity space. For example, assume that a double-bit error is detected by the WEDAC 52 in the code word retrieved from row 2 of the DRAM 30. From the code table of Fig. 5A, it can be seen that each bit of a code word stored in row 2 affects only three of the five column check bits: ~~CCB0, CCB1 and CCB4~~ BCB0, BCB1 and BCB4. In order to fully verify the parity of, for example, just ~~CCB4~~ BCB4, then it is sufficient to examine the bits stored in only three of the other seven rows: row 4, row 5, and row 7. If, in the course of retrieving and checking the code words in these rows, no other word-wise multi-bit error is detected (remember that all single-bit errors can be immediately

corrected by the WEDAC 52), then sufficient information is already available in the BEDAC 54 to unambiguously identify the two bits in error in the code word retrieved from row 0. Similarly, it can be shown that all single occurrences of double-bit errors can be resolved by scrubbing around one-half of the parity space. On the other hand, since it is not assured that a second multi-bit error will not be detected in the minimal scrub space, the sequencer 78 should perform the panic scrub as if a second multi-bit error will indeed be detected, and then, if the assumption proves incorrect, terminate the scrub early, as soon as the minimal scrub space has been examined. The information necessary to determine for each code word which other code words comprise the minimal scrub space is embodied, for example, in each of the respective ~~CCBS~~ BCB[i] columns in Fig. 5B, and can easily be extracted using convention hardware techniques. In general, it can be shown that, for larger parity spaces and well-balanced encoding algorithms, the minimal scrub space approaches one-third of the parity space.